# There's no point improving the implementation of a bad idea

devblogs.microsoft.com/oldnewthing/20070625-00

June 25, 2007

Raymond Chen

`IsBadXxxPtr` is a bad idea and you shouldn't call it. In the comments, many people proposed changes to the function to improve the implementation. But what's the point? `IsBadXxxPtr` is just a bad idea. There's no point improving the implementation of a bad idea.

On the other hand, some people suggested making it clear that `IsBadXxxPtr` is a bad idea by making it **worse**. While this is tempting in a "I'm forcing you to do the right thing" sense, it carries with it serious compatibility problems.

There's a lot of code that uses `IsBadXxxPtr` even though it's a bad idea, and making `IsBadXxxPtr` worse would risk breaking those programs that managed to get away with it up until now. The danger of this is that people would upgrade to the next version of Windows and their program would stop working. Who do you think the blame will be placed on?

Sure, you might tell these people, "That's because it's a bug in your program. Go contact the vendor for an update." Of course, that's assuming you can prove that the reason why the program stopped working was this `IsBadXxxPtr` stuff. How can you tell that that was the problem? Maybe it was caused by some other problem, possibly even a bug in Windows itself. Or is your answer just going to be "Any program that crashes must be crashing due to misuse of `IsBadXxxPtr`?"

And, as I've noted before, contacting the vendor may not be enough. Most large corporations have programs that run their day-to-day operations. Some of them may have been written by a consultant ten years ago. Even if they have the source code, they may not have the expertise, resources, or simply inclination go to in and fix it. This happens more often than you think. To these customers, the behavior change is simply a regression.

Even if you have the source code and expertise, fixing the problem may not be as simple as it looks. You may have designed your program poorly and relied on `IsBadXxxPtr` to cover for your failings. For example, you may have decided that "The `lParam` to this message is a pointer to a `CUSTOMER` structure, or it could just be the customer ID number. I'll use `IsBadReadPtr`, and if the pointer is bad, then the value must be the customer ID number."

Or you may have changed the definition of a function parameter and now need to detect whether your caller is calling the "old function" or the "new one". Or it could simply be that once you remove the call to `IsBadXxxPtr`, your program crashes constantly because the `IsBadXxxPtr` was covering up for a huge number of other programming errors (such as uninitialized variables).

"But what if I'm just using it for debugging purposes?" For debugging purposes, allow me to propose the following drop-in replacement functions:

```
inline BOOL IsBadReadPtr2(CONST VOID *p, UINT_PTR cb)
{
  memcmp(p, p, cb);
  return FALSE;
}
inline BOOL IsBadWritePtr2(LPVOID p, UINT_PTR cb)
{
  memmove(p, p, cb);
  return FALSE;
}
```

It's very simple: To see if a pointer is bad for reading, **just read it** (and similarly writing). If the pointer is bad, the read (or write) will raise an exception, and then you can investigate the bad pointer at the point it is found. We read from the memory by comparing it to itself and write to the memory by copying it to itself. These have no effect but they do force the memory to be read or written. Of course, this trick assumes that the compiler didn't optimize out the otherwise pointless "compare memory to itself" and "copy memory to itself" operations. (Note also that the replacement `IsBadWritePtr2` is not thread-safe, since another thread might be modifying the memory while we're copying it. But then again, the original `IsBadWritePtr` wasn't thread-safe either, so there's no loss of amenity there.)

(As an aside: I've seen people try to write replacements for `IsBadXxxPtr` and end up introducing a bug along the way. There are many corner cases in this seemingly-simple family of functions.)

Raymond Chen

**Follow**